psc.m

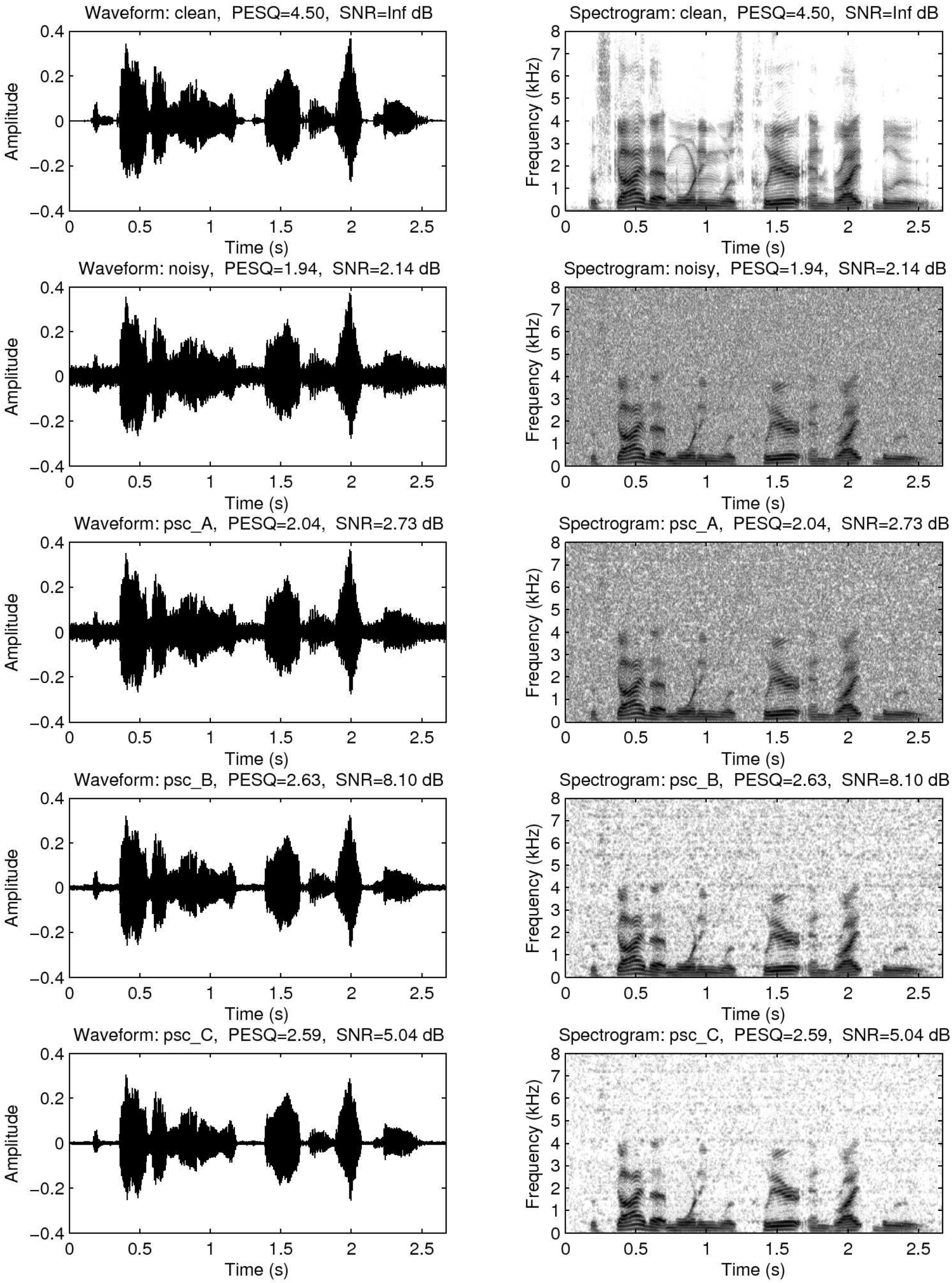
Kamil Wojcicki

March 21, 2011

Usage:

1. Start Matlab
2. Run demo by typing: `test_psc`
3. Get help by typing: `help psc`

```
%-----------------------------------------------------------------------------------------------------------------------
% Test framework for phase spectrum compensation (PSC) method for speech enhancement by Kamil Wojcicki, 2011 (test_psc.m)
clear all; close all; % clc;

    SNR = @(x,y) (10*log10((sum(x.^2)))/(sum((x(:)-y(:)).^2))); % in-line function for SNR computation

    file.clean = 'sp10.wav'; % specify the input file
    file.noisy = 'sp10_white_sn10.wav'; % specify the input file
    [speech.clean, fs, nbits] = wavread(file.clean); % read audio samples from the input file
    [speech.noisy, fs, nbits] = wavread(file.noisy); % read audio samples from the input file
    time = [0:length(speech.noisy)-1]/fs; % create time vector

    Tw = 32; % analysis frame duration (ms)
    Ts = Tw/8; % analysis frame shift (ms)
    lambda = 3.74; % scale of compensation

    % enhance noisy speech using the PSC method
    [speech.psc_A] = psc(speech.noisy, fs, Tw, Ts, 'G&L', lambda-3);
    [speech.psc_B] = psc(speech.noisy, fs, Tw, Ts, 'G&L', lambda);
    [speech.psc_C] = psc(speech.noisy, fs, Tw, Ts, 'G&L', lambda+3);

    methods = fieldnames(speech); % treatment names
    M = length(methods); % number of treatments

    %system(sprintf('rm -f ./%s.txt', mfilename));
    diary(sprintf('%s.txt', mfilename)); diary on;
    fprintf('\n%12s     %4s  %4s\n', 'Method', 'PESQ', 'SNR');
    for m = 1:M % loop through treatment types and compute SNR scores
        method = methods{m};
        mos.(method) = pesq(speech.clean, speech.(method), fs);
        snr.(method) = SNR(speech.clean, speech.(method));
        fprintf('%12s :   %4.2f  %4.2f\n', method, mos.(method), snr.(method));
    end
    diary off;

    figure('Position', [20 20 800 210*M], 'PaperPositionMode', 'auto', 'Visible', 'on');
    for m = 1:M % loop through treatment types and plot spectrograms
        method = methods{m};

        subplot(M,2,2*m-1); % time domain plots
        plot(time,speech.(method),'k-');
        xlim([min(time) max(time)]);
        title(sprintf('Waveform: %s,  PESQ=%0.2f,  SNR=%0.2f dB', method, mos.(method), snr.(method)), 'interpreter', 'none');
        xlabel('Time (s)');
        ylabel('Amplitude');

        subplot(M,2,2*m); % spectrogram plots
        myspectrogram(speech.(method), fs);
        set(gca,'ytick',[0:1000:16000],'yticklabel',[0:16]);
        title(sprintf('Spectrogram: %s,  PESQ=%0.2f,  SNR=%0.2f dB', method, mos.(method), snr.(method)), 'interpreter', 'none');
        xlabel('Time (s)');
        ylabel('Frequency (kHz)');
    end
    print('-depsc2', '-r250', sprintf('%s.eps', mfilename));
    print('-dpng', sprintf('%s.png', mfilename));

    for m = 1:M % loop through treatment types and write audio to wav files
        method = methods{m};
        audio.(method) = 0.999*speech.(method)./max(abs(speech.(method)));
        wavwrite(audio.(method), fs, nbits, sprintf('%s.wav',method));
    end

%-----------------------------------------------------------------------------------------------------------------------
% EOF
```


Waveform: clean, PESQ=4.50, SNR=Inf dB
Spectrogram: clean, PESQ=4.50, SNR=Inf dB
Waveform: noisy, PESQ=1.94, SNR=2.14 dB
Spectrogram: noisy, PESQ=1.94, SNR=2.14 dB
Waveform: psc_A, PESQ=2.04, SNR=2.73 dB
Spectrogram: psc_A, PESQ=2.04, SNR=2.73 dB
Waveform: psc_B, PESQ=2.63, SNR=8.10 dB
Spectrogram: psc_B, PESQ=2.63, SNR=8.10 dB
Waveform: psc_C, PESQ=2.59, SNR=5.04 dB
Spectrogram: psc_C, PESQ=2.59, SNR=5.04 dB
Amplitude
Frequency (kHz)
Time (s)

```
%------------------------------------------------------------------------------------------------------------------------------
% Phase Spectrum Compensation (PSC) [1] by Stark et al. Implementation by Kamil Wojcicki, 2011 (psc.m)
%
% [1] A.P. Stark, K.K. Wojcicki, J.G. Lyons and K.K. Paliwal,
%     "Noise driven short time phase spectrum compensation procedure for speech enhancement",
%     Proc. INTERSPEECH 2008, Brisbane, Australia, pp. 549-552, Sep. 2008.
%
% [2] K.K. Wojcicki, M. Milacic, A. Stark, J.G. Lyons and K.K. Paliwal,
%     "Exploiting conjugate symmetry of the short-time Fourier spectrum for speech enhancement",
%     IEEE Signal Processing Letters, Vol. 15, pp. 461-464, 2008.
%
% @inputs       s       - time domain noisy speech signal samples as a vector
%               fs      - sampling frequency (Hz)
%               Tw      - frame duration (ms)
%               Ts      - frame shift (ms)
%               stype   - overlap add synthesis type ('Allen & Rabiner', 'Griffin & Lim', 'Vanilla')
%               lambda  - strength of phase spectrum compensation (see [1])
%
% @output       y       - PSC enhanced speech signal
%
% @usage        y = psc( s, fs, Tw, Ts, stype, lambda )
%
% @example      noisy = wavread( 'sp10_white_sn10.wav' );
%               enhanced = psc( noisy, fs, 32, 32/4, 'Griffin & Lim', 3.74 );
%------------------------------------------------------------------------------------------------------------------------------
function [ y ] = psc( s, fs, Tw, Ts, stype, lambda )


    %--------------------------------------------------------------------------------------------------------------------------
    if(nargin<6), error(sprintf('Not enough input arguments. Type "help %s" for usage help.', mfilename)); end;

    %--------------------------------------------------------------------------------------------------------------------------
    s = s(:).'-mean(s);                           % make sure input signal is in row form and zero-mean
    Nw = round(fs*Tw*0.001);                      % frame duration (in samples)
    Ns = round(fs*Ts*0.001);                      % frame shift (in samples)
    nfft = 2^nextpow2(2*Nw);                      % FFT analysis length

    winfunc = @(L,S)(sqrt(2*S/(L*(2*0.5^2+(-0.5)^2)))*(0.5-0.5*cos((2*pi*((0:L-1)+0.5))/L)));
    w = winfunc(Nw, Ns);                          % Griffin & Lim's modified Hanning window

    %--------------------------------------------------------------------------------------------------------------------------
    Tn = 120;                                     % initial noise estimate: time (ms)
    M = floor(Tn*0.001*fs/Nw);                    % initial noise estimate: # of frames
    indf = Nw*[0:(M-1)].';                        % frame indices
    inds = [1:Nw];                                % sample indices in each frame
    refs = indf(:,ones(1,Nw)) + inds(ones(M,1),:);   % absolute sample indices for each frame
    frames = s(refs) * diag(w);                   % split into non-overlapped frames using indexing (frames as rows)
    S = fft(frames, nfft, 2);                     % perform short-time Fourier transform (STFT) analysis
    N = sqrt(mean(abs(S).^2));                    % estimate noise magnitude spectrum
    %N = filter( ones(1,20)/20,1,N ); N(2:nfft/2) = N(end:-1:nfft/2+2); % apply smooting?

    %--------------------------------------------------------------------------------------------------------------------------
    D = mod(length(s), Ns);                       % we will add Nw-D zeros to the end
    G = (ceil(Nw/Ns)-1)*Ns;                       % we will add G zeros to the beginning
    s = [zeros(1,G) s zeros(1,Nw-D)];             % zero pad signal to allow an integer number of segments
    L = length(s);                                % length of the signal for processing (after padding)
    M = ((L-Nw)/Ns)+1;                            % number of overlapped segments
    indf = Ns*[0:(M-1)].';                        % frame indices
    inds = [1:Nw];                                % sample indices in each frame
    refs = indf(:,ones(1,Nw)) + inds(ones(M,1),:);   % absolute sample indices for each frame
    frames = s(refs) * diag(w);                   % split into overlapped frames using indexing (frames as rows), apply analysis window
    S = fft(frames, nfft, 2);                     % perform short-time Fourier transform (STFT) analysis

    % TODO: incorporate a noise estimation algorithm ...
    A = [0, repmat(lambda,1,nfft/2-1), 0, repmat(-lambda,1,nfft/2-1)].*N; % phase spectrum compensation function
    MSTFS = abs(S).*exp(j*angle(S+repmat(A,M,1)));   % compensated STFT spectra


    %--------------------------------------------------------------------------------------------------------------------------
    x = real(ifft(MSTFS, nfft, 2));               % perform inverse STFT analysis
    x = x(:, 1:Nw);                               % discard FFT padding from frames

    switch(upper(stype))
    case {'ALLEN & RABINER','A&R'}                % Allen & Rabiner's method

        y = zeros(1, L); for i = 1:M, y(refs(i,:)) = y(refs(i,:)) + x(i,:); end; % overlap-add processed frames
        wsum = zeros(1, L); for i = 1:M, wsum(refs(i,:)) = wsum(refs(i,:)) + w; end; % overlap-add window samples
        y = y./wsum;                              % divide out summed-up analysis windows

    case {'GRIFFIN & LIM','G&L'}                  % Griffin & Lim's method

        x = x .* w(ones(M,1),:);                  % apply synthesis window (Griffin & Lim's method)
        y = zeros(1, L); for i = 1:M, y(refs(i,:)) = y(refs(i,:)) + x(i,:); end; % overlap-add processed frames
        wsum2 = zeros(1, L); for i = 1:M, wsum2(refs(i,:)) = wsum2(refs(i,:)) + w.^2; end; % overlap-add squared window samples
        y = y./wsum2;                             % divide out squared and summed-up analysis windows

    case {'VANILLA'}
        y = zeros(1, L); for i = 1:M, y(refs(i,:)) = y(refs(i,:)) + x(i,:); end; % overlap-add processed frames

    otherwise, error(sprintf('%s: synthesis type not supported.', stype));
    end

    y = y(G+1:L-(Nw-D));                          % remove the padding


%------------------------------------------------------------------------------------------------------------------------------
% EOF
```

```
% -----------------------------------------------------------------------
%                            Disclamer
%
%   This p-coded version of the PESQ measure is provided here
%   by permission from the authors. The m-file sources for the
%   PESQ measure can be found in [1].
%
%   You can also visit authors website at:
%   http://www.utdallas.edu/¬loizou/speech/software.htm
%
% -----------------------------------------------------------------------
%              PESQ objective speech quality measure
%
%   This function implements the PESQ measure based on the ITU standard
%   P.862 [2].
%
%   Usage:  pval=pesq(clean, enhanced, fs)
%
%         clean    - clean speech vector
%         enhanced - enhanced speech vector
%         fs       - sampling frequency
%         pval     - PESQ value
%
%    Note that the PESQ routine only supports sampling rates of 8 kHz and
%    16 kHz [2]
%
%  References:
%   [1] Loizou, P. (2007). Speech Enhancement: Theory and Practice,
%       CRC Press, Boca Raton: FL.
%   [2] ITU (2000). Perceptual evaluation of speech quality (PESQ), and
%       objective method for end-to-end speech quality assessment of
%       narrowband telephone networks and speech codecs. ITU-T
%       Recommendation P. 862
%
%   Authors: Yi Hu and Philipos C. Loizou
%
% Copyright (c) 2006 by Philipos C. Loizou
% Revision : 0.0  Date : 10/09/2006
% -----------------------------------------------------------------------
```